

GEOMETRIA COMPUTACIONAL

PEDRO GIGECK FREIRE 10737136

LISTA 7

6 MODIFIQUE o Algoritmo INCREMENTAL de tal forma que a sua complexidade de tempo seja reduzida para, no pior caso, $O(n \lg n)$.

O Algoritmo incremental tem complexidade $O(nh)$, porque para todos os pontos fazemos operações que consomem tempo $O(h)$.

Então temos que MODIFICAR ESSAS OPERAÇÕES para consumirem tempo $O(\lg h)$. Ambas as operações fazem buscas lineares no fecho convexo: primeiro para verificar a pertinência e depois para buscar os "Pontos de inflexão".

Vamos TROCAR essas buscas por buscas binárias (BB).

A CASCA DO algoritmo continua a mesma.

INCREMENTAL (X, Y, n)

se $\text{Esq}(x, y, 1, 2, 3)$

então $H[1] \leftarrow 1 \quad H[2] \leftarrow 2 \quad H[3] \leftarrow 3$

senão $H[1] \leftarrow 1 \quad H[2] \leftarrow 3 \quad H[3] \leftarrow 2$

para $k \leftarrow 4$ até n faça

se não $\text{PertenceBB}(H, h, X[k], Y[k])$

então $\text{InserPoint}(H, h, X, Y, k)$

devolva (H, h)

Para verificar se um ponto pertence ao fecho, ao invés de verificar todas as arestas, vamos fazer uma coisa um pouco mais complicada:

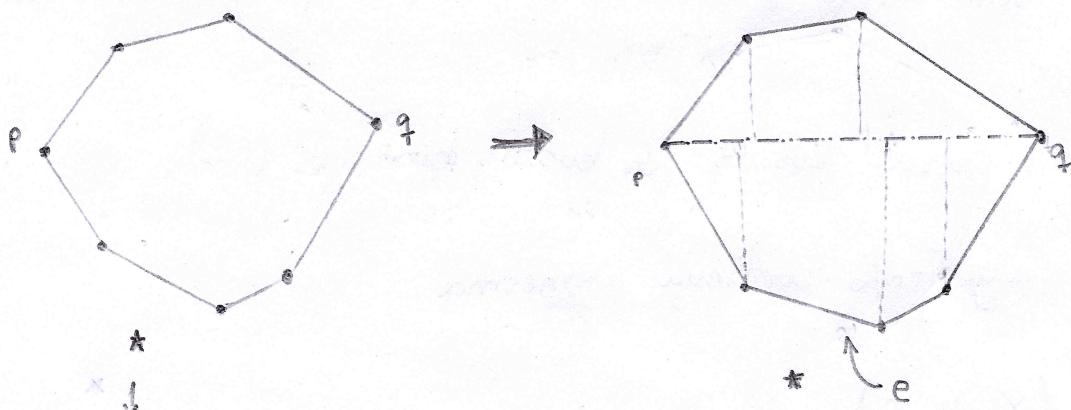
1º Achar os dois extremos do fecho atual p e q .

(com uma busca binária)

2º Dividimos o fecho na metade (cima e baixo) e desenhamos faixas verticais correspondentes a cada aresta.

3º Buscamos em qual faixa o ponto se encaixa e confermos se ele está a direita apenas dessa aresta 'e'.

POR Exemplo:



Ponto que
estamos verificando

PERTENCEBB (H, h, x, y, k)

$p \leftarrow \text{Busca.Esq}(H, h, x, y)$

$q \leftarrow \text{Busca.Dir}(H, h, x, y)$

se $\text{Esg}(x, y, H[p], H[q], k)$

então $e \leftarrow \text{Busca.Faixa}(H, h, p, q, x, y, k, \text{cima})$

senão $e \leftarrow \text{Busca.Faixa}(H, h, p, q, x, y, k, \text{baixo})$

devolva $\text{Esg}(x, y, e.p1, e.p2, k)$

Agora temos que implementar essas buscas binárias.

BuscaEsq (H, h, X, Y)

passo $\leftarrow \lceil h/2 \rceil$

atual $\leftarrow 1$

$H[0] \leftarrow H[h]$ $H[h+1] \leftarrow H[1]$ // pra não precisar do mod

enquanto $X[H[\text{atual}+1]] < X[H[\text{atual}]]$ ou

$X[H[\text{atual}-1]] < X[H[\text{atual}]]$ faça

se $Y[H[\text{atual}-1]] < Y[H[\text{atual}]]$

então $\text{atual} \leftarrow (\text{atual} + \text{passo}) \bmod h$

senão $\text{atual} \leftarrow (\text{atual} - \text{passo} + h) \bmod h$

passo $\leftarrow \lceil \text{Passo}/2 \rceil$

devolva atual

Para o BuscaDir é só trocar os ' $<$ ' por ' $>$ ' no enquanto.

Não é difícil ver que essa função funciona como uma busca binária. Se estamos na parte de baixo do fecho, olhamos só pra metade de trás (sentido horário), se estamos em cima, olhamos para metade da frente (sentido anti horário).

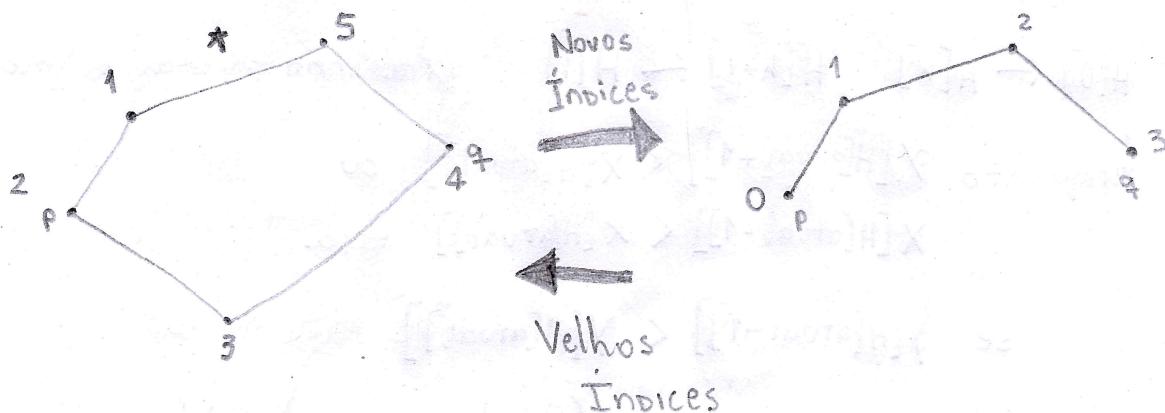
Então, em cada iteração diminuímos o espaço de busca no meio, consumindo um tempo total $O(\lg h)$.

Para fazer o BuscaFaixa, vamos fazer uma espécie de reordenação do fecho, colocando o ponto mais a esquerda p na posição H e o ponto mais a direita no 'final'.

Entre eles, apenas os pontos da metade que queremos considerar (cima/baixo).

Dessa forma, conseguiremos fazer uma busca bem mais simplificada, sem nos preocupar com os índices círculos do feixe.

Por exemplo:



Novo Índice ($H, h, p, q, \text{lado}, i$)

se ($\text{lado} = \text{cima}$ e $p < q \leq i$) ou ($\text{lado} = \text{baixo}$ e $p > q \geq i$)
 então devolva $h - |p - i|$
 senão devolva $|p - i|$

Velho Índice ($H, h, p, q, \text{lado}, i$)

se $\text{lado} = \text{cima}$
 então devolva $(h + p - i) \bmod h$
 senão devolva $(p + i) \bmod h$

Para facilitar, vamos abreviar as funções por **Novo Índice(i)** e **Velho Índice(i)**.

Na Faixa ($H, h, p, q, X, Y, K, i, \text{lado}$)

devolva $X[K] < X[H[\text{velho Índice}(i)]]$ e
 $X[K] > X[H[\text{velho Índice}(i-1)]]$

Vamos abreviar por **Na Faixa (K, i)**

BuscaFaixa ($H, h, p, q, X, Y, K, \text{lado}$)

se $X[K] > X[H[q]]$

então se lado = baixo

então devolva $(H[q-1], H[q])$

senão devolva $(H[q], H[q+1])$

se $X[K] < X[H[p]]$

então se lado = baixo

então devolva $(H[p+1], H[p])$

senão devolva $(H[p], H[p-1])$

$i \leftarrow \text{NovoIndice}(q)$

enquanto não NaFaixa(K, i)

se $X[H[\text{velhoIndice}(i)]] > X[K]$

então $i \leftarrow \lceil i/2 \rceil$

senão $i \leftarrow i + \lceil i/2 \rceil$

se lado = cima

então devolva $(H[\text{velhoIndice}(i)], H[\text{velhoIndice}(i-1)])$

senão devolva $(H[\text{velhoIndice}(i-1)], H[\text{velhoIndice}(i)])$

Com o mesmo argumento, vemos que o laço interno é uma busca binária no fecho, consumindo tempo $O(\lg h)$

UFA!

Agora, precisamos fazer o InseriPonto() em $O(\lg h)$.

Vamos usar a mesma ideia do Pertence BB, dividindo nosso espaço de busca pela metade a cada iteração

Inseri Ponto (H, h, X, Y, K)

► Acha o sentinela de cima

$$\text{parro} \leftarrow \lceil h/2 \rceil$$

$$i \leftarrow 1$$

enquanto não ($\text{Dir}(X, Y, H[i-1], H[i], K)$ e
 $\text{Esq}(X, Y, H[i], H[i+1], K)$) faça

se $\text{Dir}(X, Y, H[i], H[i+1], K)$

$$\text{então } i \leftarrow i + \text{parro}$$

$$\text{senão } i \leftarrow i - \text{parro}$$

$$\text{parro} \leftarrow \lceil \text{parro}/2 \rceil$$

► Sentinela de baixo

$$\text{parro} \leftarrow \lceil h/2 \rceil$$

$$j \leftarrow 1$$

enquanto não ($\text{Esq}(X, Y, H[j-1], H[j], K)$ e
 $\text{Dir}(X, Y, H[j], H[j+1], K)$) faça

se $\text{Esq}(X, Y, H[j], H[j+1], K)$

$$\text{então } j \leftarrow j + \text{parro}$$

$$\text{senão } j \leftarrow j - \text{parro}$$

Reorganiza Fecho (H, h, i, j, X, Y, K)

Conseguimos Encontrar os pontos de inflexão em tempo $O(\lg h)$.

Porém, para incrementar o novo ponto no fecho temos que mudar toda a estrutura do vetor, o que é custoso.

Então, temos que pensar em uma estrutura de dados que consiga inserir e deletar pontos em tempo $O(\lg h)$, ao mesmo tempo que mantém uma ordem.

Assim, seria interessante guardar o fecho em uma ABBB.

Porém, isso complicaria nosso acesso aleatório para fazer buscas binárias, o que me faz pensar em uma skip list para o fecho.

Conseguimos aceder os elementos sequencialmente ✓

Conseguimos aceder os elementos que distam de um certo parâmetro ✓

Conseguimos inserir e deletar elementos em tempo esperado $O(\lg h)$.

Como eu já fiz todo o exercício para a estrutura original, deixo
rei apenas essa explicação informal ✎.

Resumindo TUDO:

- Trocar o Fecho em VETOR por uma skip list.
 - Fazer buscas binárias no fecho, ao invés de sequenciais
- ↳ Isso nos dá um algoritmo com tempo esperado $O(n \lg n)$

8 Argumente que o QuickHull consome tempo $O(nh)$.

Seja $T(n)$ o consumo de tempo do QuickHull

Vimos em aula que

$$T(n) = T(n_d) + T(n_e) + \Theta(n)$$

Onde $n_d = r - q - 1$, quantidade de elementos na direita

$n_e = q - p' - 1$, elementos a esquerda

$n = r - p - 1$, total de elementos.

Para simplificar, assumimos uma certa linearidade de $T(n)$,

para que duas instâncias menores sejam equivalentes a uma maior.

Então assumimos que

$$T(n_d) + T(n_e) \leq T(n_d + n_e)$$

O que faz sentido, já que sabemos que $T(n) \in O(n^2)$,
então quanto maior a entrada, muito maior a saída.

Agora, vemos que $n_d + n_e = r - q - 1 + q - p' - 1 = r - p' - 2$

A parte do vetor $[p'..r]$ são os pontos que fazem parte do fecho
convexo, então $n_d + n_e \leq h$.

Portanto

$$T(n) = T(n_d) + T(n_e) + \Theta(n) \leq T(n_d + n_e) + \Theta(n) \leq T(h) + \Theta(n)$$

Como $T(n) \in O(n^2)$, então $T(h) \in O(h^2)$,

$$T(n) \leq O(h^2) + \Theta(n) \leq O(nh) + \Theta(n) = O(nh).$$

Portanto, o consumo de tempo do QuickHull é $O(nh)$.